AUTOMATED HELP SYSTEM FOR REFERENCE INFORMATION

RELATED APPLICATIONS

This application is a divisional application of U.S. Patent Appln. No. 09/191,757, filed November 13, 1998, entitled "Automatic Help System for Reference Information."

This application is related to U.S. Patent No. 6,305,008, filed Nov. 13, 1998, entitled "Automatic Statement Completion."

This application is also related to the following commonly assigned copending U.S. patent applications:

"Dynamic Parsing," U.S. Patent Appln No. 09/191,499, filed Nov. 13, 1998; and

"Indexing and Searching Across Multiple Sorted Arrays," U.S. Patent Appln. No. 09/192,057, filed Nov. 13, 1998.

COPYRIGHT NOTICE/PERMISSION

FIELD

This invention relates generally to software development environments, and more particularly to automatically providing help information.

BACKGROUND

Over time computer programs and the software programming languages used to develop them have become more complex. Computer programs are typically composed of many different source code files and programming libraries. These libraries include system libraries, networking libraries and utility libraries comprising many different functions or methods. In addition, object oriented languages have implemented a concept referred to as function overloading. Function overloading occurs when multiple functions (or methods) within a class hierarchy share the same name (or identifier), but have differing numbers of parameters or differing parameter types. Because of the proliferation of libraries and classes, the number of functions available to a software developer has steadily risen. This makes it very difficult if not impossible for a software developer to remember the calling sequence for a particular function.

In addition to function definitions, classes in object oriented languages typically have member attributes such as functions and properties. These attributes are used to define varying aspects of the class. Often the source code defining these attributes has comments associated with the attributes indicating how they are used. The number of these attributes in any single class can grow quite large, and combined with the fact that attributes can be inherited from parent classes can make it difficult for a software developer to remember the purpose and use for a particular attribute.

A further factor complicating the software development effort is the fact that it is often the case that a software module will define a large number of identifiers. These identifiers comprise typedefs, variables, macros, parameters, namespaces, templates, attributes, etc. and must each have a type, declaration and/or definition specified. It is often difficult for a developer to remember the type and identifier for these entities. In addition this information is context dependent.

One manner by which compilers and interpreters have become somewhat easier to use is through the use of the Integrated Development Environment (IDE). These environments typically support some kind of on-line help mechanism allowing a developer to refer to on-line documentation describing varying function definitions, thereby providing an improvement over printed manuals and text searching across multiple source files. Also, these environments typically have browsers capable of

opening multiple source files, allowing a developer to refer to the source file defining a particular function or class attribute while editing another source file. In addition, for object oriented languages, the IDE may also provide a browser allowing the developer access to the class hierarchy enabling a developer to browse a class definition. While on-line help and browsers are an improvement to printed manuals and text searching, several problems remain. First, in order to look up the definition of a function or member attribute, the developer must locate the source file containing the function or attribute definition, consult the on-line help file, or locate the class name in the class hierarchy browser.

Second, the user must locate the function or attribute definition in the file, which typically involves either scrolling through the file, using a text search capability to search through the file, or using the browser to locate the function or attribute definition in the file. Finally, once the function or attribute definition has been located, the user must typically alternate back and forth between the window containing the definition of the function or attribute (either in a file window or a class hierarchy window) and the window containing the source code currently being edited.

SUMMARY

The above-identified problems, shortcomings and disadvantages with the prior art, as well as other problems, shortcoming and disadvantages, are solved by the present invention, which will be understood by reading and studying the specification and the drawings. In one embodiment, the system includes an editor to provide for developing source code for a computer program. The source code includes statements containing identifiers. In addition, the editor, upon detecting an event from a pre-determined set of events, invokes an automatic help module. The automatic help module displays reference information associated with the identifier.

Thus one aspect of the invention is that while a programmer is developing or writing source code for a program, the automatic help module is invoked upon the occurrence of a predetermined event. The event can be the positioning of a cursor over an identifier followed by a clicking a mouse button, hovering the mouse cursor over an

identifier, selecting a menu or icon after highlighting the identifier, or the event can be the entry of identifier into the source code. The automatic help module then displays reference information regarding the identifier. In one embodiment of the invention, the reference information is a list of parameters for the function with the associated data type for each parameter. The first required parameter in the displayed list is highlighted. As the developer enters further function parameters (if any) in the source code, the next required parameter is highlighted. In addition, in one embodiment of the invention, as the user provides more data, the list of overloaded functions and their associated parameter sets can be pruned to remove those items that are incompatible with what the user has already entered.

In a further alternative embodiment, the source code is searched for comments associated with an identifier. If any comments are found, the comments are displayed, thereby providing potentially useful information to the developer. Thus embodiments of the invention have advantages not found in prior systems. The developer is able to view relevant reference information regarding functions and attributes without having to consult on-line help directories, source code files containing identifier definitions, paper documentation, or other potential sources of identifier information. This saves the developer time and does not interrupt the developer's train of thought. In this manner, development of computer programs is made easier and more productive as compared to previous systems. The invention includes systems, methods, computers, and computer-readable media of varying scope. Besides the embodiments, advantages and aspects of the invention described here, the invention also includes other embodiments, advantages and aspects, as will become apparent by reading and studying the drawings and the following description. In an alternative embodiment of the invention, the reference information is the data type of the identifier.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a diagram of the hardware and operating environment in conjunction with which embodiments of the invention may be practiced;

FIG. 2 shows a block diagram of a system according to one embodiment of the invention;

FIG. 3(a) shows a sample C++ class definition;

FIGs. 3(b), 3(c), 3(d), 3(e) and 3(f) are illustrations of representative screen shots of an IDE where the automatic help module has been invoked; and

FIG. 4 shows a flowchart illustrating a method according to one embodiment of the invention.

DETAILED DESCRIPTION

In the following detailed description of exemplary embodiments of the invention, reference is made to the accompanying drawings that form a part hereof, and in which is shown by way of illustration specific exemplary embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized and that logical, mechanical, electrical and other changes may be made without departing from the spirit or scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims.

The detailed description is divided into four sections. In the first section, the hardware and the operating environment in conjunction with which embodiments of the invention may be practiced are described. In the second section, a system of one embodiment of the invention is presented. In the third section, a method, in accordance with an embodiment of the invention, is provided. Finally, in the fourth section, a conclusion of the detailed description is provided.

Hardware and Operating Environment

Referring to FIG. 1, a diagram of the hardware and operating environment in conjunction with which embodiments of the invention may be practiced is shown. The description of FIG. 1 is intended to provide a brief, general description of suitable computer hardware and a suitable computing environment in conjunction with which the invention may be implemented. Although not required, the invention is described in the general context of computer-executable instructions, such as program modules, being executed by a computer, such as a personal computer. Generally, program modules

include routines, programs, objects, components, data structures, etc., that perform particular tasks or implement particular abstract data types.

Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCS, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

The exemplary hardware and operating environment of FIG. 1 for implementing the invention includes a general purpose computing device in the form of a computer 20, including a processing unit 21, a system memory 22, and a system bus 23 that operatively couples various system components include the system memory to the processing unit 21. There may be only one or there may be more than one processing unit 21, such that the processor of computer 20 comprises a single central-processing unit (CPU), or a plurality of processing units, commonly referred to as a parallel processing environment. The computer 20 may be a conventional computer, a distributed computer, or any other type of computer; the invention is not so limited.

The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory may also be referred to as simply the memory, and includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system (BIOS) 26, containing the basic routines that help to transfer information between elements within the computer 20, such as during start-up, is stored in ROM24. The computer 20 further includes a hard disk drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media.

The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive

interface 33, and an optical disk drive interface 34, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for the computer 20. It should be appreciated by those skilled in the art that any type of computer-readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs), and the like, may be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24, or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37, and program data 38. A user may enter commands and information into the personal computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the monitor, computers typically include other peripheral output devices (not shown), such as speakers and printers.

The computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as remote computer 49. These logical connections are achieved by a communication device coupled to or a part of the computer 20; the invention is not limited to a particular type of communications device. The remote computer 49 may be another computer, a server, a router, a network PC, a client, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 20, although only a memory storage device 50 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local-area network (LAN) 51 and a wide-area network (WAN) 52. Such networking environments are commonplace in office networks, enterprise-wide computer networks, intranets and the Internet, which are all types of networks.

When used in a LAN-networking environment, the computer 20 is connected to the local network 51 through a network interface or adapter 53, which is one type of communications device. When used in a WAN-networking environment, the computer 20 typically includes a modem 54, a type of communications device, or any other type of communications device for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It is appreciated that the network connections shown are exemplary and other means of and communications devices for establishing a communications link between the computers may be used.

The hardware and operating environment in conjunction with which embodiments of the invention may be practiced has been described. The computer in conjunction with which embodiments of the invention may be practiced may be a conventional computer, a distributed computer, or any other type of computer; the invention is not so limited. Such a computer typically includes one or more processing units as its processor, and a computer-readable medium such as a memory. The computer may also include a communications device such as a network adapter or a modem, so that it is able to communicatively couple other computers.

System

In this section of the detailed description, a description of a computerized system according to an embodiment of the invention is provided. The description is provided by reference to FIG. 2. Referring now to FIG. 2, a system according to an embodiment of the invention includes an Integrated Development Environment (IDE) 200. As shown, the IDE200 includes an editor 205, parser 215, automatic help module 220 and database 225. Those of ordinary skill within the art will appreciate that the IDE 200 also may include other components, not shown in FIG. 2; only those parts necessary to describe the invention in an enabling manner are provided. The parser 215 may be a parser for any type of programming language; the invention is not so limited. For example, the parser 215 can in different embodiments parse the C, C++, Pascal, Visual BASIC, or Java

programming languages etc., all of which are known in the art. In addition, IDE 200 may include what is known in the art as an interpreter.

The parser 215 converts source code 210 into executable code (not shown in FIG. 2). The source code 210 is a text description of a computer program, as written in a given programming language by or for one or more computer programmers. Typically and as is known in the art, the source code comprises a series of statements defining the data structures and the actions the computer is to perform using the data structures. These statements are composed of various programming language tokens, which are combined to form declarations and definitions that describe the entities that make up the computer program. Identifiers are used to identify particular entities in the program, such as function names, variable names, class names, macro names and template names. Those of ordinary skill in the art will recognize that various entities and identifier mechanisms are used in various programming languages.

The executable code is that which is produced by the IDE 200, so that the computer program can actually be run on a computer (for example, a computer as has been described in the preceding section of the detailed description). The parser 215 operates to parse the source code 210 according to the parsing rules applicable to a particular programming language. In one embodiment of the invention, the parser 215 stores and retrieves information to and from database 225.

The editor 205 of the IDE 200 provides for the developing (writing) of the source code 210 of a computer program.

Automatic help module 220 is invoked by editor 205 upon the occurrence of an event and employs the method described below with reference to FIG. 4 to automatically provide reference information on an identifier in the source code.

Database 225 is typically a file comprising a database that, in one embodiment of the invention, is used by the parser to store information including, but not limited to class definitions, member data types, member function names and their associated parameters. In addition, reference information such as source file names and line numbers where an entity is defined or referenced is stored in database 225. Database 225 typically includes information not only from source code 210, but also includes information from other sources including system header files and files from class libraries including Microsoft

Foundation Class (MFC) header files and ActiveX Template Libraries (ATL), all of which are known in the art. The database 225 stores an indicator as to which of the above-mentioned sources were used to populate the particular database record.

In one embodiment of the invention, database 225 is referred to as an NCB (No Compile Browse) file, and is populated and maintained by the parser module 215. In this embodiment, the parser module 215 dynamically updates the database 225 as the source code is modified or added to by the user. The creation and maintenance of the database 225 by parser module 215 is more fully described in commonly assigned, cofiled and copending application entitled "Dynamic Parsing" which has been previously incorporated by reference.

In an alternative embodiment of the invention, database 225, while appearing as one database to the user, is actually comprised of multiple stores or databases. In this embodiment, a first database is dynamically updated by the parser, as described above. In addition to the first dynamically updated database, one or more pre-created databases exist. The pre-created databases contain information that seldom changes, such as operating system declarations and header files, Microsoft Foundation Class definitions and header files, and the ActiveX Template Library referred to above. Those of ordinary skill in the art will recognize that other class definitions and header files could be included in the pre-created database. It is desirable to provide such a pre-created database because of the large amount of information that is provided by the class definitions and header files. This information seldom changes, and therefore does not need to be re-parsed and stored in the dynamically updated database. This allows the parser to dynamically parse the user developed code, which does change frequently, in an acceptable amount of time.

Thus, in accordance with one embodiment of the invention, the system of FIG. 2 operates as follows. A developer drafts the source code 210 using the editor 205. As the developer is writing the source code 210, editor 205 detects a predetermined event and invokes automatic help module 220. Automatic help module 220 then queries database 225 and uses the information found therein to display a tooltip box providing reference information on an identifier in the source code. In one embodiment of the invention, the predetermined event occurs when a user enters a particular token or type of token.

Examples of tokens include identifier names, function name, class names, macro names, global and local variable names, and operators.

In an alternative embodiment, the automatic help module 220 is invoked when the user positions a cursor over an identifier. The module can be invoked either by pressing a button, or by allowing the mouse cursor to "hover" over the token.

In a further alternative embodiment of the invention, automatic help module 220 is invoked when the user highlights a token and presses a mouse button or hot-key. A hot-key is known in the art, and typically comprises a function key or sequence of predefined keys that have special meaning to an application.

The invention is not limited to the mechanisms described above to invoke the automatic help module, and those of ordinary skill in the art will appreciate that other mechanisms can be utilized and are within the scope of the invention. Such mechanisms include menu selection and icon selection FIG. 3(a) presents a sample class definition and FIGs. 3(b) - 3(f) illustrate representative screen shots of editing sessions using the class definition. FIGs. 3(a) - 3(f) are discussed in the context of the C++ programming language, however the invention is not so limited. The invention is adaptable to any programming language. Referring now to FIG. 3(a), a C++ class definition 350 is presented for the class named "Foo". Sample class Foo has three member attributes, an x coordinate location attribute 355, a y coordinate location attribute 360, and a z coordinate location attribute 365. The three attributes have comments associated with them indicating their intended use. In addition, class Foo has a member function getVolume 370 which accepts three parameters. The declaration for getVolume is preceded by a code comment indicating the method's purpose. The class definition for class Foo may be contained in the source code 210 file currently being edited, or more commonly, it may be contained in any of a number of files comprising the source code for the application.

Referring now to FIG. 3(b), a block diagram of a representative screen shot of an editor component of an IDE according to one embodiment of the invention is presented.

Within screen 300 of display 302, several previously entered lines of code 304 are shown, along with a current line of code 306. As the developer is editing line 306, the developer enters an expression component comprising object pointer name, "m_pfoo", which is an identifier that points at an object of class Foo and a reference to class member

function"getVolume" followed by a `( `. The editor detects a possible entry of the member function identifier because of the `("token, and invokes the automatic help module 220. Automatic help module 220 invokes the parser 215 to parse the code from the start of the function to the expression at the cursor position and returns parser data that is used by the automatic help module220 to create a tooltip box 308 to display the function type, class name, and parameter list for the function getVolume. Because the user has not entered any parameters yet, the first parameter is considered a missing required parameter and is highlighted. In the representative screen shot, highlighting is indicated by displaying the parameter in bold type. Those of ordinary skill in the art will recognize that other highlighting mechanisms could be used, such as italic fonts, alternate text colors, underlining or any combination thereof. As the user enters parameters, the highlighting is moved to the next missing parameter. In an alternative embodiment of the invention (not shown), code comments are displayed in tooltip box 308. In the example above, the function return type and parameter set shown in tooltip box 308 are displayed in addition 5 to the text "//getVolume calculates a volume for the object."

The parser action described above is more fully described in the cofiled, copending, coassigned applications entitled "Dynamic Parsing" and "Automatic Statement Completion", both of which have been previously incorporated by reference.

In FIG. 3(c), a screen of an alternative embodiment of the invention is shown. Similar to screen 300, screen 316 contains a section of previously entered lines of code304. Current code line 322 starts with an identifier "m_pfoo", which is a pointer to a member of the class Foo. The identifier is followed by the C/C++ pointer operator. Drop down box 324 is displayed by an automatic statement completion module that has been more fully described in cofiled, coassigned and copending application entitled"Automatic Statement Completion", previously incorporated by reference. The drop down box displays a list of valid tokens that can follow the pointer operator. As the user moves the cursor down the drop down box, automatic help module 220 displays tool tip box 330 next to the highlighted entry, which contains help information about the identifier. In one embodiment of the invention, comments within the source code near the declaration of the entity, if any, are displayed. In a further embodiment (not shown), entity data type information is displayed.

It is desirable to display the help information only if the user pauses at an entry. In this manner, the help is provided unobtrusively and only as required so that the user's actions are not delayed and the user's train of thought is not unduly interrupted. In addition, it is desirable that the system utilizes only idle CPU cycles to provide the help information. Idle cycles are typically plentiful, especially while the programmer pauses to think over their code.

In some programming languages, function overloading makes it possible for a single function identifier to be used for multiple function definitions, each having a unique set of parameters and parameter data types. In this case, an embodiment of the invention displays a drop-down box similar to drop down box 324 which includes a line for each overloaded function, in addition to other valid tokens that may be entered at that point in the current statement. As the user moves down the box, automatic help module 220 displays a tool tip box 330 containing help information for the highlighted entry in the drop down box. For the functions in the list, the help information includes the parameter list for the highlighted entry. In an alternative embodiment of the invention, as the user enters parameters, the list in drop down box 324 is pruned to remove those functions where the entered parameter types do not match the particular overloaded function's parameter set.

In FIG. 3(d), a screen shot of an alternative embodiment of the invention is shown. In this embodiment, screen 318 contains previously entered code 304, current line 322, and drop down box 324, all described with reference to FIG. 3(c) above. Here, the user has highlighted an entry in the drop down box corresponding to the "y" member attribute of class Foo. In this embodiment of the invention, tool tip box X26 contains comments located on or near the line in the source code where the "y" member attribute is defined (source line 365 of FIG. 3(a) in this example). Automatic help module 220 queries database 225 in order to obtain the information necessary to locate the source code line where "y" is defined.

FIG. 3(e) shows a representative screen shot of an alternative embodiment of the invention where data type information is presented to the user. Like screen 318, screen 332 contains previously entered code 304. In addition, screen 332 contains a current line 323. In screen 332, the user has entered the "z" member attribute of class Foo. In this

embodiment of the invention, tooltip box 340 displays help information to the user comprising the data type of the identifier, and the class, if any, to which the identifier belongs. Automatic help module 220 queries database 225 to obtain the data type and class information for the identifier.

Referring now to FIG 3(f), a representative screen shot of an alternative embodiment of the invention is presented. Like screens 318 and 332, screen 320 of display 302 contains several previously entered lines of code 304. In addition, screen 316 has a current line of code 312. As the developer is editing line 312, the developer enters an expression component comprising object pointer name, "m_pfoo", which is an identifier that points at an object of class Foo and a reference to member attribute "x".

The IDE detects the entry of the member attribute identifier, and invokes the automatic help module 220. Automatic help module 220 causes tooltip box 314 to display the comments associated with the attribute "x". In an alternative embodiment of the invention, automatic help module 220 causes tooltip box 314 to be displayed when a mouse cursor is positioned near the "x" in line 312. In a further alternative embodiment the control key down as the mouse cursor hovers near the identifier.

As discussed above, the reference information contained in the tooltip boxes of the various embodiments of the invention include data type, class membership, and code comments associated with a token or identifier. Those of ordinary skill in the art will recognize that other types of reference information could be displayed and are within the scope of the invention. For example, information obtained from on-line documentation resident on the computer system or available from another system connected via a network could also be displayed in the tooltip box.

In the embodiments described above with reference to FIGs. 3(b) - 3(f), the automatic help module uses the parser to determine the data types and classes for the tokens when help information is displayed. For example, the "m_pfoo" identifier shown in line 322 is a pointer to a member of the class Foo. The automatic help module 220 must use the parser to discover the fact that m_pfoo is such a pointer, and that it refers to a member of class Foo. The example expression provided in line 322 is a relatively simple example used to illustrate the invention, however the invention is not so limited, and the expressions for which help information is provided can be arbitrarily complex.

The operation of the parser described above is more fully described in the patent application entitled "Dynamic Parsing" and "Automatic Statement Completion" which have been previously incorporated by reference.

Thus, in this manner, the invention provides for advantages not found within the prior art. As the source code is written (developed) by a computer programmer via the editor, the automatic help module is invoked upon the occurrence of an event to provide reference information for an identifier. Unlike prior systems and methods, the developer does not have to take focus away from the editor to consult paper documentation, on-line documentation, other source files, or other tool windows containing class hierarchies and the like.

Method

In this section of the detailed description, a method according to an embodiment of the invention is presented. This description is provided in reference to FIG. 4. The computerized method is desirably realized at least in part as one or more programs running on a computer -- that is, as a program executed from a computer-readable medium such as a memory by a processor of a computer. The programs are desirably storable on a computer-readable medium such as a floppy disk or a CD-ROM, for distribution and installation and execution on another (suitably equipped) computer. Thus, in one embodiment, a computer program is executed by a processor of a computer from a medium therefrom to automatically provide help in the form of reference information on an identifier.

Referring now to FIG. 4, a flowchart of a method according to one embodiment of the invention is shown. In 400, at least a section of source code for a computer program is developed (written). Such source code may be written by a computer programmer utilizing an editor component of an IDE. In 405, help information comprising reference, information on an identifier is displayed upon the occurrence of an event. In one embodiment, the display of the help information in 405 is accomplished via 410, 412, 415, 420 and 425.

In 410, a predetermined event is detected indicating that automatic help 20 information should be displayed. Typically, an editor component of an IDE will detect

the event, however other components of the BCE may also detect the event. In one embodiment, the event occurs when the developer enters an identifier for a function (method) or attribute. In an alternative embodiment, the event occurs when a cursor is positioned over the identifier. In a further alternative embodiment of the invention, the event occurs when the mouse cursor "hovers" over the identifier. In yet another embodiment of the invention, the event occurs when the user selects from a menu, selects an icon, or uses a context sensitive popup menu.

Upon the detection of an event at 410, the method proceeds to 412 and invokes a parser to determine the data type and class, if any, to which the identifier belongs. The parser applies the rules applicable to the particular programming language in order to determine the data type and class. The operation of the parser is more fully described in the patent application entitled "Dynamic Parsing" and "Automatic Statement Completion", which have been previously incorporated by reference.

The method then proceeds to 415, which uses the type and class information obtained from step 412 as the basis for a query to search for information on the identifier. In one embodiment of the invention, a dynamic database containing class definition information from potentially multiple varying sources is searched. Function definition information, source file names and line numbers where the identifier is defined are included in the information stored in the database. In one embodiment of the invention, if the identifier is a function name, the parameter list for the function is retrieved from the database. A parameter list is also obtained for macros, templates, attributes and other entities that have a parameter list. In the case of an overloaded function, a plurality of parameter lists are obtained. If the identifier is an attribute name, the file name containing the attribute definition and the line number within the file where the attribute is defined are retrieved from the database. The method then uses the information to scan the file for comments near the attribute definition. These comments are then retrieved from the source file.

At 420 a check is made to see if any information in the form of data from a database or code comments was retrieved at 415. If insufficient information is available, the method terminates. Otherwise, the method proceeds to 425 where the information is displayed. In one embodiment of the invention, a tooltip box displays the information

retrieved at 415. In one embodiment of the invention, if the identifier is a function name, and the function is what is known in the art as an overloaded function, then a drop-down box containing a plurality of function parameter sets is displayed. In an alternative embodiment of the invention, as the user enters more information, the list of items in the drop-down box is pruned to remove those functions that are incompatible with what the user has entered so far. The reference information displayed for the function reflects the parameters for the remaining functions in a tooltip box.

In an embodiment of the invention, the first parameter in the parameter list displayed in the tooltip box is highlighted. As the developer enters further parameters in the source code, the highlighting moves to the next parameter to be supplied by the developer. When all required parameters have been supplied, the help information is no longer displayed.

In another embodiment of the invention, while one instance of automated help is at work, if another event causing automated help is detected the automated help is provided for the second instance and upon completion the previous instance is resumed. This is especially useful if one of the parameters to a function is also a function (referred to as a second function). In this case, parameter help for the second function is provided and when that is finished, parameter help for the original function is resumed. In yet another embodiment of the invention, when the user hovers over an identifier (or an event is detected that invokes `identifier info' i.e. selection of an icon, menu item, or context sensitive popup menu) then automatic help is provided about that identifier using the mechanism outlined above.

Conclusion

An automated help system has been described. Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement which is calculated to achieve the same purpose may be substituted for the specific embodiments shown. This application is intended to cover any adaptations or variations of the present invention. Therefore, it is manifestly intended that this invention be limited only by the following claims and equivalents thereof.